

# REPAST MODEL TESTING GUIDE

JONATHAN OZIK, NICK COLLIER - REPAST DEVELOPMENT TEAM

## 0. BEFORE WE GET STARTED

Before we can do anything with Repast Symphony, we need to make sure that we have a proper installation of Repast Symphony 2.1. Instructions on downloading and installing Repast Symphony on various platforms can be found on the Repast website.

## 1. GETTING STARTED WITH REPAST SIMPHONY MODEL TESTING

Developing a useful agent-based simulation involves acquiring sufficient knowledge of a models domain, developing the conceptual model, often while grounding it in more abstract theories of the domain, and then translating the model into software. The work can be complex and is rarely a waterfall type process where each step is completed before the next step begins. Rather, the process is an iterative one where the development of the model may require additional domain knowledge that, in turn, requires changes in the software. Furthermore, model specifications and requirements may change also requiring changes in the software. Test-driven development (TDD) is a general software development technique that can help with managing this complexity. In TDD the development of software is driven by the creation of many small automated unit tests each of which exercise some small part of the larger system's behavior. An aspect of the extreme-programming movement that has gained popularity since the late 1990s, the technique itself gained in prominence with the publication of Kent Beck's seminal work "Test-Driven Development: By Example"<sup>1</sup>.

The present guide will walk you through a number of agent-model TDD use cases with Repast Symphony. For those interested in learning more about agent-model testing, including the benefits of TDD when developing agent models, see Collier and Ozik (2013)<sup>2</sup>. For more information on the JUnit testing framework that we will be using, see <http://junit.org>.

## 2. AGENT-MODEL TESTING USE CASES WITH REPAST SIMPHONY

To add tests into an existing Repast Symphony project, we recommend the following setup steps:

---

*Date:* July 10, 2013.

<sup>1</sup>Beck, K. 2003. Test Driven Development: By Example. Boston, Massachusetts: Addison-Wesley.

<sup>2</sup>Collier, N, and J Ozik. Test-Driven Agent-Based Simulation Development. To appear in WSC 2013 Proceedings. Washington, D.C., 2013.

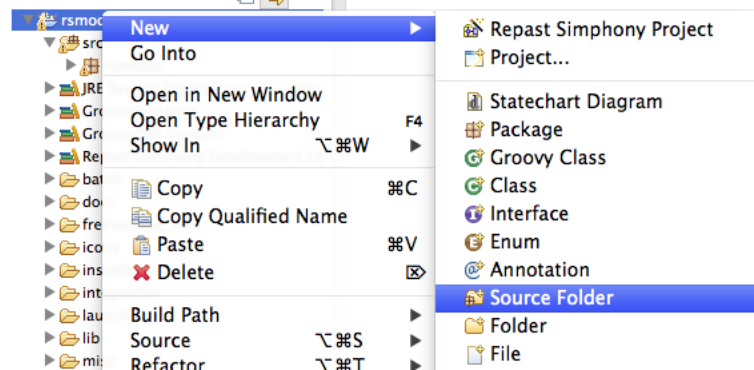


FIGURE 1. Selecting your project, right-clicking and choosing *New* → *Source Folder*.

- (1) Add a **test** source folder to the project. This can be accomplished in a number of ways. One way is to right click on the project and navigate to *New* → *Source Folder* (Fig. 1). Then fill in **test** in the *Folder name* text field and click on *Finish* (Fig. 2).
- (2) Modify the output folder for the **test** source folder. Right click on the project and navigate to *Properties*. Then choose the *Java Build Path* entry in the left bar and select the *Source* tab. Select the check box that reads *Allow output folders for source folders* and expand the **test** entry (Fig. 3). Select the *Output folder* entry and click on the *Edit* button (Fig. 4). Then select the *Specific output folder* option and input **testbin**, or something else different from **bin** (Fig. 5). Click on *Okay* and then *Okay* again.
- (3) Add a JUnit Test Case by right clicking on the project in the Package Explorer view, choosing *New* → *Other...* (Fig. 6). Then navigate to *Java* → *JUnit* and choose *JUnit Test Case* (Fig. 7)<sup>3</sup>. In the *New JUnit Test Case* wizard choose the *New JUnit 4 test* option, specify the correct source folder (**test**), name your test case, include all the method stubs, and click on *Finish* (Fig. 8)<sup>4</sup>. If JUnit was not previously on your project's build path, you will see a dialog asking if you'd like to add it (Fig. 9). Click *OK* to add the JUnit 4 library to the project's build path.

<sup>3</sup>This creates a Java JUnit test case, but a Groovy JUnit test case can be used as well.

<sup>4</sup>While this document is focused on JUnit 4, JUnit 3 or other testing frameworks can also be utilized.

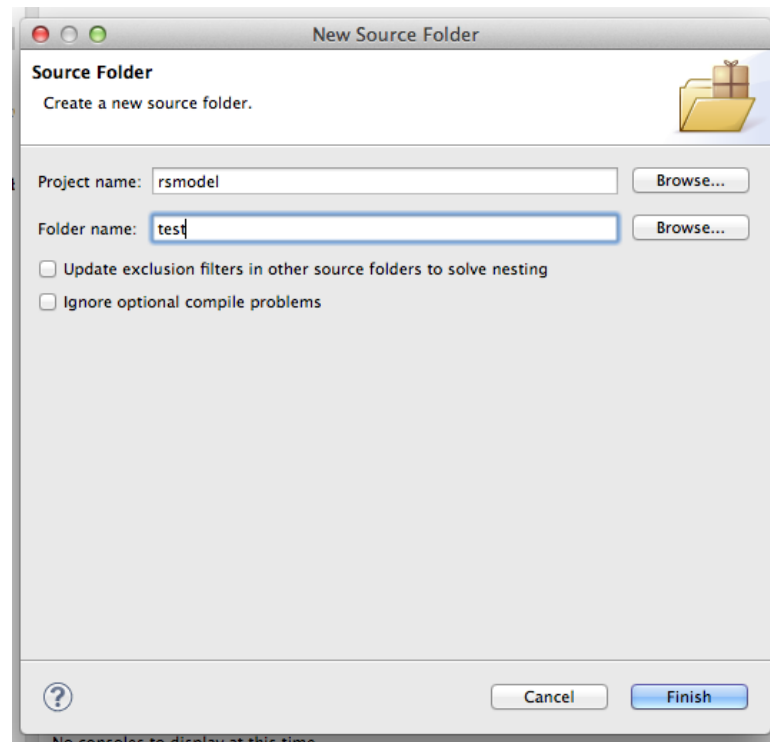


FIGURE 2. New source folder wizard.

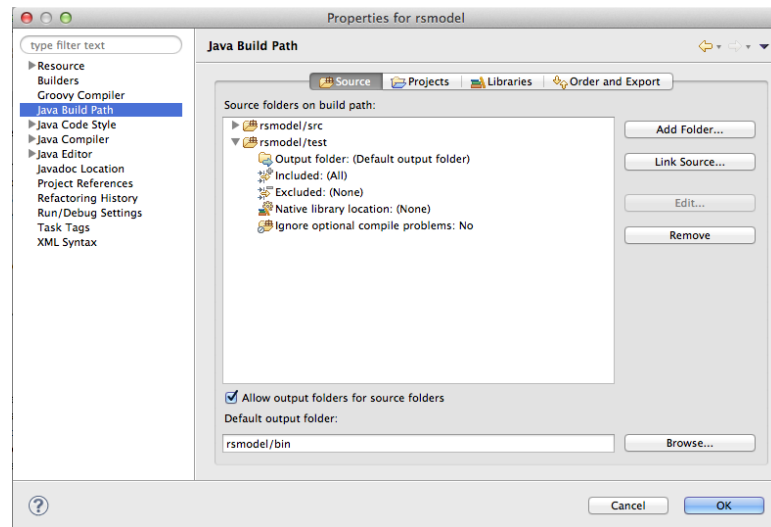


FIGURE 3. *Java Build Path* → *Source* tab in a project's properties.

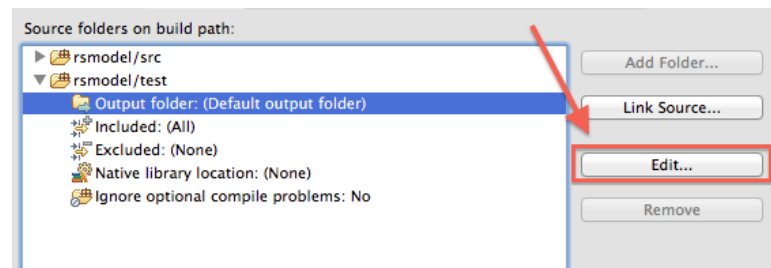


FIGURE 4. Edit the output folder of the `test` source folder.

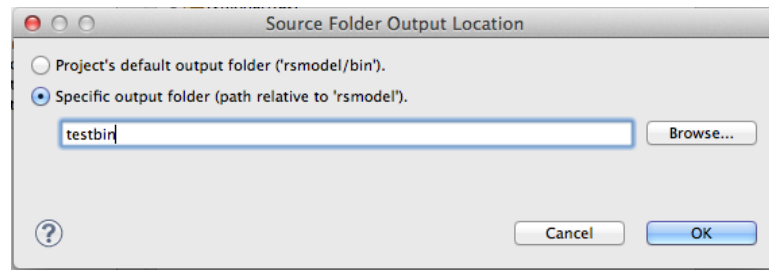


FIGURE 5. Choosing the source folder output location.

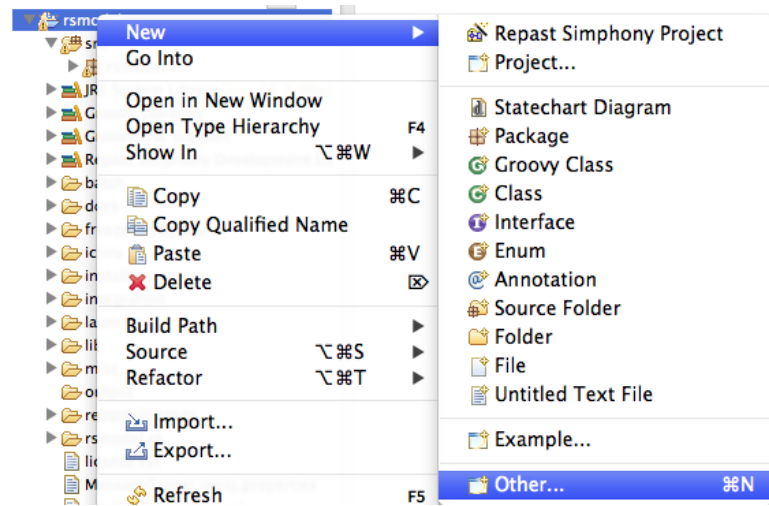


FIGURE 6. Selecting your project, right-clicking and choosing New → Other... .

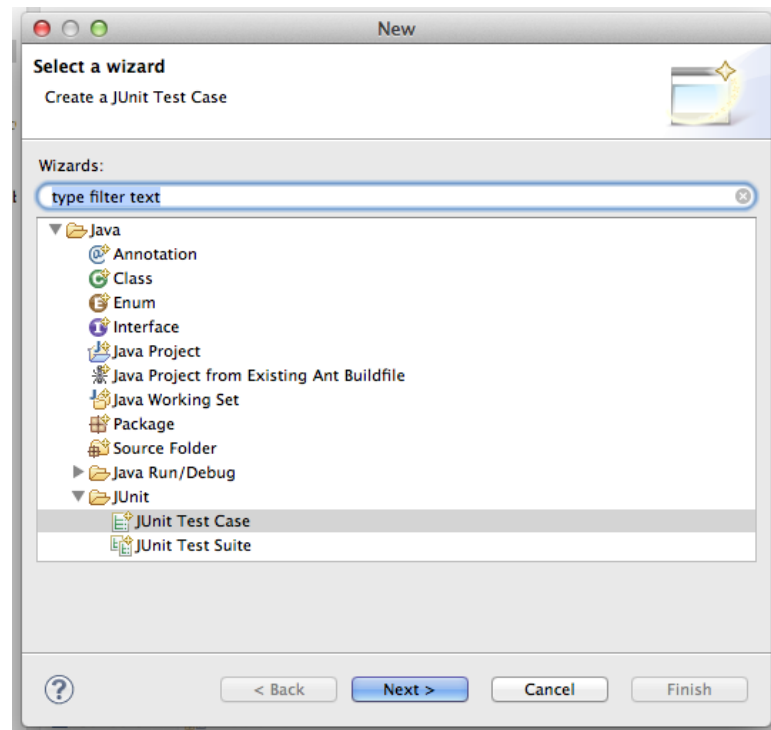


FIGURE 7. The JUnit Test Case option within Java → JUnit.

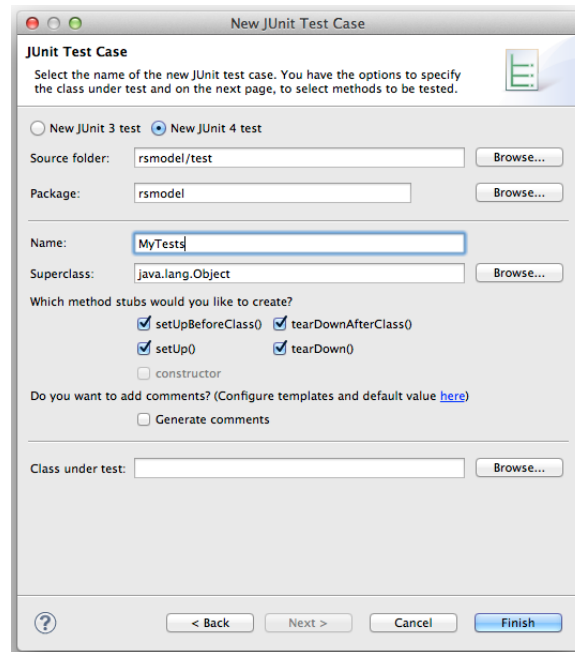
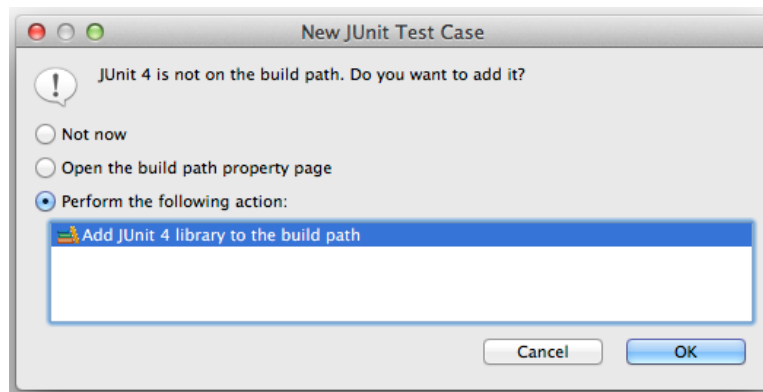
FIGURE 8. The new *JUnit Test Case* wizard.

FIGURE 9. Add JUnit 4 library to build path.

Once the setup is complete, there are a number of different types of model tests that can be created, based on the nature of the model behavior that is being tested. We will go over a few such cases next.

**2.1. Use Case 1: Simple Unit Testing with Repast Symphony Models.** If the elements being tested are relatively decoupled, there is nothing special that needs to be done in terms of test case setup. In this scenario, the `@BeforeClass`, `@AfterClass`, `@Before`, and `@After` annotated methods do not need any Repast Symphony specific elements and tests can be written in the usual way JUnit tests are written.

**2.2. Use Case 2: Schedule Based Model Testing with Repast Symphony Models.** Some model behaviors that you might want to test involve scheduled behaviors. For example, you might want to know that at a certain tick, specific model actions had occurred. This is especially relevant to any Statecharts related behaviors. In this situation, the `RunEnvironment` will need to be passed a `Schedule` object. Listing 1 shows an example where the schedule based setup is executed before each of the tests that will be run (i.e., in the `@Before` annotated method).

```

1  @Before
2  public void setUp() throws Exception {
3      Schedule schedule = new Schedule();
4      RunEnvironment.init(schedule, null, null, true);
5      Context context = new DefaultContext();
6      RunState.init().setMasterContext(context);
7
8      // Any additional setup
9  }
```

LISTING 1. `@Before` setup method in a schedule dependent test case.

Line 3 in Listing 1 shows the creation of the `Schedule` object. It is then sent as the first parameter to the `RunEnvironment` static `init` method. The second and third parameters can be `null` for this case and the fourth parameter indicates whether this is a batch (i.e., headless) run, which it is so we specify `true`. Line 5 creates a new `DefaultContext` that is set as the master context of the `init`-ed `RunState` in line 6. Lines 5 and 6 are only strictly necessary for testing behaviors that rely on a master context being set, which is the case for Statecharts.

At this point, tests can be written such as in Listing 2:



```

1  @Test
2  public void testUninfectedToInfected() {
3      ISchedule schedule = RunEnvironment.getInstance().
4          getCurrentSchedule();
5      Person p = new Person();
6      assertEquals(UNINFECTED, p.getStatus());
7      for (int i = 0; i < 5; ++i) {
8          schedule.execute();
9      }
10     assertEquals(INFECTED, p.getStatus());
11 }

```

LISTING 2. Example test method where the schedule is advanced.

In this hypothetical example, a Person agent is created, it is verified that the person's status is UNINFECTED, the schedule is advanced 5 times, at which point the person's status is checked to see that it is INFECTED. An important item to note here is that the scheduler in Repast Symphony doesn't just allow for discrete time steps so the fact that the schedule is executed 5 times doesn't necessarily mean that we will find ourselves at tick 5 after the for loop, unless there were actions scheduled only to occur on every tick.

**2.3. Use Case 3: Context Builder Based Model Testing with Repast Symphony Models.** For cases where the specific setup defined in a ContextBuilder is required for testing, the @Before testing setup might look like Listing 3.

```

1  . . .
2  public Context context;
3
4  @Before
5  public void setUp() throws Exception {
6      context = new DefaultContext();
7      MyContextBuilder builder = new MyContextBuilder();
8      context = builder.build(context);
9
10     // Any additional setup
11 }
12 . . .

```

LISTING 3. @Before setup method in a context builder dependent test case.

Here, after creating the DefaultContext, the context builder is used to build it into the state it should be in at the start of each simulation run. A test utilizing the setup in Listing 3 might look like Listing 4. In this hypothetical test, a new Person agent is added

to the main context and a `hasNeighbors` method is called to ensure that the added `Person` agent has neighbors in the pre-built context.

```
1  @Test
2  public void testAddingPersonToContext() {
3      Person p = new Person();
4      context.add(p);
5      assertTrue(p.hasNeighbors());
6  }
```

LISTING 4. Example test method where the context state created by a context builder is used.

**2.4. Use Case 4: Model Testing with ReLogo Models.** ReLogo models come with the infrastructure and associated assumptions of the ReLogo world which is built by the `SimBuilder` context builder included with each model. To test a ReLogo model there are a few additional steps needed for the test setup beyond what was done in Section 2.3. Listing 5 shows the setup (optionally) separated into `@BeforeClass` and `@Before` components. The idea here is that the ReLogo world is built once before all the tests are run but before each individual test is executed, the observer clears the ReLogo world state, removing existing turtles and links and resetting patches.

```

1  . . .
2  static UserObserver observer;
3
4  @BeforeClass
5  public static void setUpBeforeClass() throws Exception {
6      String scenarioDirString = "ModelName.rs";
7      ScenarioUtils.setScenarioDir(
8          new File(scenarioDirString).getAbsoluteFile());
9      File paramsFile = new File(ScenarioUtils.getScenarioDir(),
10         "parameters.xml");
11      ParametersParser pp = new ParametersParser(paramsFile);
12      Parameters params = pp.getParameters();
13      RunEnvironment.init(new Schedule(), null, params, true);
14      Context context = new DefaultContext();
15      SimBuilder builder = new SimBuilder();
16      context = builder.build(context);
17
18      // If statecharts are used in the ReLogo model
19      RunState.init().setMasterContext(context);
20
21      observer = (UserObserver)context.iterator().next();
22
23      // Any additional before class setup
24  }
25
26  @Before
27  public void setUp() throws Exception {
28      observer.clearAll();
29
30      // Any additional setup
31  }
32  . . .

```

LISTING 5. @Setup method in a schedule dependent test case.

Some of the contents of Listing 5 will already look familiar from the previous sections but there are additions that we expand on here:

- (1) The scenario directory (the .rs folder) is specified in Line 6. Replace “Model-Name.rs” with the name of the scenario directory in your project.
- (2) The parameters.xml file within the scenario directory is parsed and passed to the RunEnvironment init method. This can be done in non-ReLogo models as well.
- (3) If statecharts are used in the ReLogo model, Line 18 should be included as well.